Characterizing Covers of Functional Dependencies using FCA

Victor Codocedo[1], Jaume Baixeries[2], Mehdi Kaytoue[1], and Amedeo Napoli[3]

[1] Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
[2] Universitat Politècnica de Catalunya. 08032, Barcelona. Catalonia.
[3] LORIA (CNRS - Inria Nancy Grand Est - Université de Lorraine), B.P. 239, F-54506, Vandœuvre-lès-Nancy.

Corresponding author: victor.codocedo@inria.fr

Abstract. Functional dependencies (FDs) can be used for various important operations on data, for instance, checking the consistency and the quality of a database (including databases that contain complex data). Consequently, a generic framework that allows mining a sound, complete, non-redundant and yet compact set of FDs is an important tool for many different applications. There are different definitions of such sets of FDs (usually called *cover*).
In this paper, we present the characterization of two different kinds of covers for FDs in terms of pattern structures. The convenience of such a characterization is that it allows an easy implementation of efficient mining algorithms which can later be easily adapted to other kinds of similar dependencies. Finally, we present empirical evidence that the proposed approach can perform better than state-of-the-art FD miner algorithms in large databases.

1 Introduction

Functional Dependencies (FDs) are a keystone of the relational database model, since they allow to check the consistency and maintain the quality of a database [7, 9, 8], and to guide the database design [17]. In addition, they have been used to study information integration in the Web of data with varying degrees of quality [21, 22], or to check the completeness in DBpedia [1]. Therefore, the computation of a succinct representation of a set of FDs (usually refered to as a **cover**), is of interest to various fields of knowledge discovery and representation, specially, if this computation can easily be extended to other kinds of dependencies.

The computation of *covers* for FDs is a popular topic in database literature. As a reference, in [18], seven algorithms to mine FDs and compute their covers are reviewed and grouped into three families: *lattice transversal algorithms*, *difference/agree sets algorithms*, and *dependency induction* algorithms. The characterization of FDs with FCA and pattern structures has also be presented in [2] which in turn, have been generalized to other types of FDs [5]. For a more detailed review on the characterization of FDs and FCA, see [3].

In this paper, we characterize these representations of FD covers using pattern structures, an extension of FCA dealing with complex object representations [15]. On the

one hand, in Section 3.2 we discuss on the relation between the Stem Base of implications [11] with the Minimal Cover of dependencies, a sound, complete, non-redundant set of FDs that has minimum cardinality w.r.t. any other equivalent cover. Our main contribution shows that the latter can be characterized by pseudo-extents of a partition pattern structure.

On the other hand, in Section 3.1 we adapt the definition of a canonical direct basis of implications with proper premises [4, 19] to the formalism of pattern structures, in order to prove that this basis is equivalent to a reduced non-redundant cover of FDs.

Finally, in Section 5 we empirically compare these two ways of computing a cover for a set of FDs with the algorithm TANE [14]. This algorithm is one of the most popular FD mining algorithms and according to [18], it is the base for the family of "lattice transversal algorithms" and serves as the base to compare our proposal with a state-of-the-art algorithm for computing covers for FDs.

2 Theoretical Background

Let $\mathcal{U}$ be a set of attributes, and let Dom be a set of values (a domain). For the sake of simplicity, we assume that Dom is a numerical set. A tuple t is a function $t : \mathcal{U} \mapsto Dom$, and a table T is a set of tuples. Usually a table is represented as a matrix, as in Table 1, where the set of tuples (or objects) is $T = \{ t_1, t_2, \ldots, t_7 \}$ and $\mathcal{U} = \{ a, b, c, d, e \}$ is the set of attributes. We use *table, dataset, set of tuples* as equivalent terms. We overload the functional notation of a tuple in such a way that, given a tuple $t \in T$, we say that $t(X)$ (for all $X \subseteq \mathcal{U}$) is a tuple with the values of t in the attributes $x_i \in X$:

$$t(X) = \langle t(x_1), t(x_2), \ldots, t(x_n) \rangle$$

For example, we have that $t_2(\{ a, c \}) = \langle t_2(a), t_2(c) \rangle = \langle 2, 1 \rangle$. In this article, the set notation is dropped: instead of $\{ a, b \}$ we use ab.

2.1 Functional Dependencies and their covers

Definition 1 ([20]). *Let T be a set of tuples, and $X, Y \subseteq \mathcal{U}$. A* **functional dependency (FD)** *$X \to Y$ holds in T if:*

$$\forall t, t' \in T : t(X) = t'(X) \implies t(Y) = t'(Y)$$

For instance, the functional dependency $d \to e$ holds in T, whereas the functional dependency $e \to d$ does not hold since $t_4(e) = t_5(e)$ but $t_4(d) \neq t_5(d)$.

For a given set of functional dependencies F, we can use the three Armstrong's axioms (reflexivity, augmentation and transitivity) to derive a larger set of FDs [17]. We will call F* the set of FDs derived from F by reflexivity and augmentation, and F^+ the set of FDs derived by reflexivity augmentation and transitivity. Two sets of FDs F and H are said to be equivalent $F \equiv H \iff F^+ = H^+$.

Let F be a set of FDs from a database with tuples T. F is said to be *sound* if all FDs in F hold in T. In addition, F is said to be *complete* if all FDs that hold in T can be derived from F. Let $X \to Y$ be any FD in F, then F is said to be *non-redundant* if

$F \backslash \{X \rightarrow Y\} \not\equiv F$ for any $X \rightarrow Y$. Additionally, F is said to be *non-redundant w.r.t. augmentation* iff $(F \backslash \{X \rightarrow Y\})^* \neq F^*$

A set F is said to be *left-reduced* if $\forall X \rightarrow Y \in F$ and $Z \subset X$ we have that $(F \backslash \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\} \not\equiv F$. Dually, it is said to be *right-reduced* if $\forall X \rightarrow Y \in F$ and $Z \subset Y$ we have that $(F \backslash \{X \rightarrow Y\}) \cup \{X \rightarrow Z\} \not\equiv F$. F is said to be *reduced* if it is simultaneously *left* and *right-reduced*.

Let F be a reduced set of FDs, then we can derive $G = \{X \rightarrow y \mid \forall X \rightarrow Y \in F, y \in Y\}$. G is the *splitting* of F and both sets are equivalent. Let F be a *reduced* set, its splitting is called a *canonical cover*. A *canonical cover* is a *left-reduced*, *non-redundant w.r.t. augmentation* set of FDs with a single element in their right hand side (a split set). Notice that *canonical covers* can be redundant w.r.t. transitivity. For example the *canonical cover* of Table 1 contains $\{c \rightarrow b, c \rightarrow e, d \rightarrow e, bd \rightarrow c, be \rightarrow c\}$ where $bd \rightarrow e$ is redundant w.r.t. transitivity, i.e. $bd \rightarrow bde \rightarrow c$.

Finally, a set F is said to be a *minimum cover* if it has as few FDs as any other equivalent set of FDs. For example, the *minimum cover* of Table 1 contains FDs $\{c \rightarrow be$, $d \rightarrow e, be \rightarrow c\}$. Notice that the minimum cover is not restricted to be reduced, so it is not presented with split sets. Secondly, the cardinality of the example minimum cover contains exactly one fewer FD than the canonical cover, namely $bd \rightarrow c$.

2.2 Formal Concept Analysis (FCA)

For the sake of brevity we do not provide a description of the FCA framework. The notation used in this article follows [12] where $\mathcal{K} = (G, M, I)$ is a formal context of objects G, attributes M and incidence relation I, with formal concepts (A, B).

2.3 Implication Systems

Implications are relations established between attribute sets from a formal context $\mathcal{K}$. Implications are analogous to FDs and they can be used to characterize the latter [2]. Implication systems (sets of implications) can also characterize FD covers [4].

An implication $X \rightarrow Y$ holds in $\mathcal{K}$ for $X, Y \subseteq M$ if $Y \subseteq X''$. Notice that we maintain the notation of FDs to denote the relation between X and Y, the reason for this will become apparent in what follows. Let T be a set of tuples and $\mathcal{U}$, a set of attributes in a table (such as the one in Table 1). We define the set $Pair(T) = T \times T$ as the Cartesian product of the set of tuples T and incidence set I such that $((t_i, t_j), x) \in I \iff t_i[x] = t_j[x]$, $\forall x \in \mathcal{U}$, $\forall t_i, t_j \in T$. It can be shown that an FD $X \rightarrow Y$ holds in the database if and only if $X \rightarrow Y$ is an implication of the formal context $\mathcal{K} = (Pair(T), \mathcal{U}, I)$ [12]. $\mathcal{K}$ is called the *binary codification* of table T. For example, Table 3 contains the binary codification of Table 1. In Table 3 we can observe the implication $d \rightarrow e$ which can be verified as an FD in Table 1.

The previous statement entails that FDs and implications in $\mathcal{K}$ are in 1-1 correspondence. Moreover, the corresponding definition of a canonical cover of FDs is equivalent to that of a *canonical-direct unitary basis* of implications as shown in [4].

	a	b	c	d	e
t_1	1	1	1	1	1
t_2	2	1	1	1	1
t_3	3	1	2	2	2
t_4	3	2	3	2	2
t_5	3	1	2	3	2
t_6	1	1	2	2	2
t_7	1	1	2	4	2

Table 1: Example of a table T

	a	b	c	d	e
$\delta(a)$	×				
$\delta(b)$		×			
$\delta(e)$					×
$\delta(a) \sqcap \delta(b)$	×	×	↙		↙
$\delta(a) \sqcap \delta(e)$	×				×
$\delta(d) \sqcap \delta(e)$				×	×
$\delta(a) \sqcap \delta(d) \sqcap \delta(e)$	×	↙	↙	×	×
$\delta(b) \sqcap \delta(c) \sqcap \delta(e)$		×	×		×
$\delta(a) \sqcap \delta(b) \sqcap \delta(c) \sqcap \delta(e)$	×	×	×	↙	×
$\delta(b) \sqcap \delta(c) \sqcap \delta(d) \sqcap \delta(e)$	↙	×	×	×	×
$\delta(a) \sqcap \delta(b) \sqcap \delta(c) \sqcap \delta(d) \sqcap \delta(e)$	×	×	×	×	×

Table 2: Representation context

	a	b	c	d	e
(t_1, t_2)		×	×	×	×
(t_1, t_3)		×			
(t_1, t_4)					
(t_1, t_5)		×			
(t_1, t_6)	×	×			
(t_1, t_7)	×	×			
(t_2, t_3)		×			
(t_2, t_4)					
(t_2, t_5)		×			
(t_2, t_6)		×			
(t_2, t_7)		×			
(t_3, t_4)	×			×	×
(t_3, t_5)	×	×	×		×
(t_3, t_6)		×	×	×	×
(t_3, t_7)		×	×		×
(t_4, t_5)	×				×
(t_4, t_6)				×	×
(t_4, t_7)					×
(t_5, t_6)		×	×		×
(t_5, t_7)		×	×		×
(t_6, t_7)	×	×	×		×

Table 3: Binary codification of Table 1

2.4 Partition Pattern Structures

Partition Pattern Structures (PPS) is a type of pattern structure [15] that deals with, as the name suggests, object representations in the form of set partitions. PPS have shown to be useful for mining biclusters [6] and, more importantly for this article, relations between partition pattern concepts have been used to characterize FDs of different kinds [3].

The formalization of a database with tuples T and attributes $\mathcal{U}$ as a PPS is as follows. A partition d of T is a set $\mathrm{d} \subseteq \wp(T)$ of disjoint subsets of T such that for any two different elements $K_i, K_j \in \mathrm{d}$ we have that $K_i \cap K_j = \varnothing$ and $\bigcup_{K_i \in \mathrm{d}} K_i = T$.

Let D be the set of all possible partitions of T, they can be ordered by a *coarser/finer* relation denoted $\mathrm{d}_i \sqsubseteq \mathrm{d}_j$ (d_i is finer than d_j for $\mathrm{d}_i, \mathrm{d}_j \in \mathrm{D}$) iff $\forall\, K_i \in \mathrm{d}_i, \exists\, K_j \in \mathrm{d}_j : K_i \subseteq K_j$. The similarity operator is defined as $\mathrm{d}_i \sqcap \mathrm{d}_j = \{K_i \cap K_j \mid K_i \in \mathrm{d}_i, K_j \in \mathrm{d}_j\}$.

From this, it follows that $(\mathrm{D}, \sqcap)$ is a complete lattice with supremum and infimum defined respectively as $\top = \{\{t\} \mid t \in T\}$ and $\bot = \{\{T\}\}$.

The set of attributes $\mathcal{U}$ is mapped onto $(\mathrm{D}, \sqcap)$ through a function δ which for a given attribute $x \in \mathcal{U}$ yields its equivalence relations over T. With this, we can configure the PPS $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ where (X, d) is a partition pattern concept with $X^{\square} = \mathrm{d}$ denoting the equivalence relations implied by attributes in X, and $\mathrm{d}^{\square} = X$ denoting all attributes with associated equivalence relations finer than d.

Theorem 1. *Let $(Pair(T), \mathcal{U}, I)$ be the binary codification of a table within a database, and $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ its PPS representation:*

$$(W, X) \in (Pair(T), \mathcal{U}, I) \iff (X, \mathrm{d}) \in (\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$$

The proof of Theorem 1 can be found in [3]. Theorem 1 presents an important property of the PPS that states that $X \subseteq \mathcal{U}$ is an extent in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ if and only if it is also an intent in $(Pair(T), \mathcal{U}, I)$ (the relation between the set of tuple pairs $W \subseteq Pair(T)$ and the partition $\mathrm{d} \in \mathrm{D}$ can be formalized as well but it is of no interest for our development). Theorem 1 is very important since it entails that the lattices derived from $(Pair(T), \mathcal{U}, I)$ and $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ are isomorphic. Consequently, implication $X \rightarrow Y$ in $(Pair(T), \mathcal{U}, I)$ can be found as a relation between extents in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ (extent implication) such that $Y \subseteq X^{\square\square}$.

3 Covers and Pattern Structures

In this section we present two different kinds of covers for FDs: canonical covers (Subsection 3.1) and minimal covers (Subsection 3.2), as well as their characterization in terms of Pattern Structures. This section uses existing well-known results in FCA, which are reviewed here for the sake of readability.

First, we present how a canonical cover for FDs can be computed with pattern structures, according to the results in [4, 19]. Then, in Subsection 3.2 we present a novel characterization of a minimum cover of FDs by means of pseudo-intents [12]. The interest of these results are not only limited to computing covers for FDs, but also for computing covers for generalizations of FDs.

3.1 Mining a Canonical Cover of FDs

The characterization of a canonical cover of FDs using FCA is straightforward and it has been previously studied in [4, 19]. In a nutshell, a canonical direct basis of implications with proper premises is analogous to a reduced non-redundant cover of FDs, i.e. a canonical cover. In this section we recall some of these ideas and show how they can be simply adapted to the framework of partition pattern structures.

Firstly, let $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ be a PPS, we define $\underline{\mathrm{D}} = \{\mathrm{d} \in \mathrm{D} \mid \mathrm{d}^{\square\square} = \mathrm{d}\}$ as the set of all closed partition patterns in D, then the triple $(\mathcal{U}, \underline{\mathrm{D}}, J)$ with $(\mathrm{d}, x) \in J \iff \mathrm{d} \sqsubseteq \delta(x)$ is called a representation context of $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ and their corresponding concept lattices are isomorphic [15]. For the sake of readability of the following definitions, we will prefer to define the representation context as $(\underline{\mathrm{D}}, \mathcal{U}, J)$ instead of $(\mathcal{U}, \underline{\mathrm{D}}, J)$. By transitivity of equivalence, it is clear that $(\underline{\mathrm{D}}, \mathcal{U}, J)$ is isomorphic to $(Pair(T), \mathcal{U}, I)$ as defined in Section 2.4 and as such, implications in $(\underline{\mathrm{D}}, \mathcal{U}, J)$ correspond to FDs. For example, Table 2 (not considering elements $\swarrow$) contains the representation context $(\underline{\mathrm{D}}, \mathcal{U}, J)$ of the partition pattern structure derived from Table 1. Notice that objects are closed intersections of object representations, e.g. $\delta(d)$ is not present since $\delta(d)^{\square\square} = de$ (thus $d \rightarrow e$). With this, the canonical direct basis of implications in $(\underline{\mathrm{D}}, \mathcal{U}, J)$ (and thus, canonical cover of FDs) is determined by the set of proper premises of elements in $\mathcal{U}$.

Theorem 2 ([19]). *$X \subseteq \mathcal{U} \backslash \{y\}$ is a premise of $y \in \mathcal{U}$ iff X is a hypergraph transversal of $\mathcal{H}_y^{\swarrow}$ defined as :*

$$\mathcal{H}_y^{\swarrow} = \{((\mathcal{U} \backslash \{y\})) \backslash \mathrm{d}' \mid \mathrm{d} \in \underline{\mathrm{D}}, \mathrm{d} \swarrow y\}$$

The set of all proper premises of y is exactly the minimum hypergraph transversal $Tr(\mathcal{H}_y^{\swarrow})$.

Detailed descriptions of the development of Theorem 2 can be found in [19]. Providing a formal definition of hypergraph transversals is out of the scope of this article, however we briefly mention that this formalization can also be made considering set collections (instead of hypergraphs) and minimum hitting sets (instead of minimum hypergraph transversals) [10]. Alternatively, this problem is analogous to the vertex cover problem [11].

Theorem 2 provides a formal description for the proper premises of a given attribute $y \in \mathcal{U}$ that in turn yields the canonical cover of functional dependencies. However this approach requires the creation of the representation context which creates a middle step in the overall mining process. Actually, by analyzing the arrow relation between d and y we can observe that the representation context is not really necessary.

$$\mathrm{d} \swarrow y \iff (\mathrm{d}, y) \notin J \text{ and if } \mathrm{d}' \subsetneq \mathrm{h}' \text{ then } (\mathrm{h}, y) \in J$$

We have that in $(\underline{\mathrm{D}}, \mathcal{U}, J)$, $\mathrm{d}' = \{x \in \mathcal{U} \mid (\mathrm{d}, x) \in J(\iff \mathrm{d} \sqsubseteq \delta(x))\}$ and thus $\mathrm{d}' \equiv \mathrm{d}^{\square}$ for any $\mathrm{d} \in \underline{\mathrm{D}}$. Moreover, in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$, $\mathrm{d}^{\square} \subsetneq \mathrm{h}^{\square} \iff \mathrm{h} \subsetneq \mathrm{d}$ since $\mathrm{h} = \mathrm{h}^{\square\square}$ and $\mathrm{d} = \mathrm{d}^{\square\square}$. With this, we can rewrite the previous definition as follows.

$$\begin{aligned} \mathrm{d} \swarrow y &\iff (\mathrm{d}, y) \notin J \text{ and if }, \mathrm{d}' \subsetneq \mathrm{h}' \text{ then } (\mathrm{h}, y) \in J \\ &\iff \mathrm{d} \not\sqsubseteq \delta(y) \text{ and if } \mathrm{d}^{\square} \subsetneq \mathrm{h}^{\square} \text{ then } \mathrm{h} \sqsubseteq \delta(y) \\ &\iff \mathrm{d} \not\sqsubseteq \delta(y) \text{ and if } \mathrm{h} \subsetneq \mathrm{d} \text{ then } \mathrm{h} \sqsubseteq \delta(y) \end{aligned}$$

The last result shows that $\mathrm{d} \swarrow y$ in $(\underline{\mathrm{D}}, \mathcal{U}, J)$ can be defined directly over the PPS. Intuitively, this definition corresponds to $y \nearrow d$ in $(\mathcal{U}, \underline{\mathrm{D}}, J)$ and thus, in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$. With these elements we can finally propose a characterization for the canonical cover of functional dependencies in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ as follows.

Corollary 1. *Let $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$ be a partition pattern structure and $Tr(\mathcal{H})$ denote the hypergraph transversal of $\mathcal{H}$, then with*

$$\begin{aligned} \mathcal{L}_{cc} &= \{X \to y \mid y \in \mathcal{U}, X \in Tr(\mathcal{H}_y^{\nearrow})\} \\ \mathcal{H}_y^{\nearrow} &= \{((\mathcal{U} \backslash \{y\})) \backslash \mathrm{d}' \mid \mathrm{d} \in \underline{\mathrm{D}}, y \nearrow \mathrm{d}\} \\ y \nearrow \mathrm{d} &\iff \mathrm{d} \not\sqsubseteq \delta(y) \textit{ and if } \mathrm{h} \subsetneq \mathrm{d} \textit{ then } \mathrm{h} \sqsubseteq \delta(y) \end{aligned}$$

$\mathcal{L}_{cc}$ is a canonical cover of functional dependencies.

For the running example, let us calculate the proper premises of attribute c using the arrow relations in Table 3. We have $c \nearrow (\delta(a) \sqcap \delta(b))$ and $c \nearrow (\delta(a) \sqcap \delta(d) \sqcap \delta(e))$. With this, we have the hypergraph $\mathcal{H}_c^{\nearrow} = \{\{d, e\}, \{b\}\}$ for which the minimum transversal hypergraph is $Tr(\mathcal{H}_c^{\nearrow}) = \{\{b, d\}, \{b, e\}\}$. Correspondingly, we have the FDs $bd \to c$ and $be \to c$ which are included in the canonical cover.

3.2 Characterizing a Minimum Cover of FDs with FCA

We introduce a novel characterizations of a minimum cover of FDs by means of pseudo-intents, and its generalization using pseudo-extents in a PPS.

The stem base of implications, or Duquenne-Guigues basis [13], is a *sound*, *complete* and *non-redundant* basis which also has minimum cardinality among the sets of implications for a given formal context. We show how this can be used to characterize a minimum cover of FDs in a rather simple manner. Prior to introducing the stem base, let us define pseudo-closed sets [12].

Definition 2. *(Pseudo-closed sets) Let $P \mapsto P''$ be a closure operator over a set M, then P is a pseudo-closed set if and only if ($Q \subset P \equiv Q \subseteq P$ and $Q \neq P$):*

$$P \neq P'' \tag{1}$$

$$Q \subset P \text{ and } Q \text{ is a pseudo-closed set } \implies Q'' \subseteq P \tag{2}$$

Given a formal context (G, M, I), pseudo-closed sets $A \subseteq G$ are called pseudo-extents, while pseudo-closed sets $B \subseteq M$ are called pseudo-intents. A stem base of implications, or Duquenne-Guigues basis, can be defined as follows:

Theorem 3 ([12]). *(Duquenne-Guigues Basis) The set of implications:*

$$\mathcal{L} = \{P \rightarrow P'' \mid P \text{ is a pseudo-intent}\} \tag{3}$$

is sound, complete and non-redundant.

Theorem 4 ([12]). *Every complete set of implications contains an implication $X \rightarrow Y$ with $X'' = P''$ for every pseudo-intent P of (G, M, I)*

Theorem 4 entails that the stem base of implications has minimum cardinality with respect to any equivalent set of implications of (G, M, I). With this and the previous observation that FDs are in 1-1 correspondence with implications in $(Pair(T), \mathcal{U}, I)$, we can derive the following corollary.

Corollary 2. *Let $(Pair(T), \mathcal{U}, I)$ be the formal context derived from a database with tuples T and attributes $\mathcal{U}$. The set of functional dependencies $\mathcal{L} = \{P \rightarrow P''\}$ is a minimum cover of tuples T.*

Corollary 2 provides a novel characterization of the minimum cover of functional dependencies through pseudo-intents of a formal context. At this point, we are in the position to use the FCA machinery to calculate this minimum cover, however we would like to propose a generalization of the stem base using PPS which allows for a more direct representation of the database, avoiding the creation of the costly formal context $(Pair(T), \mathcal{U}, I)$ which grows quadratically w.r.t. the number of tuples in T.

Noticeably, in the PPS $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$, we can maintain the notion of pseudo-extents for a pseudo-closed set $X \subseteq \mathcal{U}$ with $X \mapsto X^{\square\square}$.

Proposition 1. *Let $(\mathcal{U}, (D, \sqcap), \delta)$ be the PPS representation of a database, then the set of functional dependencies:*

$$\mathcal{L} = \{X \to X^{\square\square} \mid X \text{ is a pseudo-extent}\} \tag{4}$$

is a minimum cover.

The proof of Proposition 1 follows from Theorem 1 and the observation that there is a bijective mapping between extents in $(Pair(T), \mathcal{U}, I)$ and intents in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$. Moreover, for a set $X \in \mathcal{U}$ the closure operator $X \mapsto X''$ is exactly equivalent to $X \mapsto X^{\square\square}$ and consequently, the set of pseudo-intents in $(Pair(T), \mathcal{U}, I)$ is the same as the pseudo-extents in $(\mathcal{U}, (\mathrm{D}, \sqcap), \delta)$. Thus, because of Corollary 2, Proposition 1 holds.

4 Mining a Minimum Cover of FDs

We describe the mechanisms and algorithms through which we are able to implement the ideas exposed in Section 3 to build a Minimum Cover Miner.

4.1 Identifying Pseudo-Closed Sets

Pseudo-closed sets can be identified by means of pre-closed sets. For the sake of readability, we do not provide a full characterization of pre-closed sets. We simply introduce the properties that render them useful for identifying pseudo-closed sets. A thorough description of pre-closed sets can be found in [11].

Proposition 2. *Let $Q \mapsto Q''$ be a closure operator. A set Q is pre-closed if and only if it is either a pseudo-closed set or a closed set.*

Proposition 2 states that the set of pre-closed sets is composed by the union of the set of pseudo-closed sets and the set of closed sets considering given an arbitrary closure operator. The usefulness of this proposition is apparent since, given a mechanisms to identify pre-closed sets, we can simply test those that are not closed sets to identify pseudo-closed sets. Fortunately, pre-closed sets form a closure system itself and thus, this mechanisms exists and it is described in Proposition 3.

Proposition 3. *Let $\mathcal{L} = \{P \to P''\}$ be the stem base of implications, then $\mathcal{L}^\bullet(X)$ is the pre-closure of X when:*

$$\begin{aligned} X^{\mathcal{L}} &= X \cup \bigcup \{P'' \mid P \to P'' \in \mathcal{L}, P \subset X\} \\ X^{\mathcal{L}\mathcal{L}} &= X^{\mathcal{L}} \cup \bigcup \{P'' \mid P \to P'' \in \mathcal{L}, P \subset X^{\mathcal{L}}\} \\ &\dots \\ \mathcal{L}^\bullet(X) &= X^{\mathcal{L}\mathcal{L}\dots\mathcal{L}} \text{ s.t. } \mathcal{L}^\bullet(X)^{\mathcal{L}} = \mathcal{L}^\bullet(X) \end{aligned}$$

The pre-closure of a set X corresponds to a fixed point that is iteratively calculated using the stem base of implications. While it seems odd that in order to calculate the stem base of implications we need it in the first place, it is important to notice that we only need a *part* of the stem base, particularly those rules such that $P \subset X$ and $\mathcal{L}^\bullet(X)$. This is achieved by means of a *lectical* enumeration of the powerset of X.

4.2 Lectical Enumeration of Pre-Closures

Pre-closures are enumerated in lectical order using the *PreviousClosure* algorithm, a variation of *NextClosure* [11, 16]. The lectical order of the powerset of $\mathcal{U}$ lists all the subsets of a set X before listing X for any $X \subseteq \mathcal{U}$. In addition, incomparable elements w.r.t. $\subseteq$ are listed in inverted to the lexicographical order $\leqslant_l$. Thus, the *lectical* order $\prec$ between two sets $X, Y \subseteq \mathcal{U}$ is defined as:

$$X \prec Y \iff X \subseteq Y \text{ or } Y \leqslant_l X$$

For example, given $\mathcal{U} = \{a, b, c, d\}$, the lectical order of $\wp(\mathcal{U})$ is:

$$d, c, cd, b, bd, bc, bcd, a, ad, ac, acd, ab, abd, abc, abcd$$

4.3 Algorithms

Algorithm 1 depicts a brief summary of *PreviousClosure*. It receives a *partial* stem base $\mathcal{L}$ and a list of *candidates* in lectical order. For each candidate X , it calculates its pre-closure $\mathcal{L}^\bullet(X)$ using the stem-base. If the pre-closure is lexicographically greater than the candidate, then firstly, all remaining candidates that are lectically lower than the pre-closure are discarded for future enumerations (pruning), and secondly, the pre-closure is returned. When no more candidates are available, the algorithm returns *null*.

The pruning mechanism works by removing from the candidate list elements Y that would yield redundant pre-closures, i.e. elements $X \subseteq Y \subseteq \mathcal{L}^\bullet(X)$ for which $\mathcal{L}^\bullet(Y) = \mathcal{L}^\bullet(X)$. This efficient pruning mechanism is possible due to the following proposition.

Proposition 4. *Let $X \subseteq Y$ and $X \leqslant_l Y$, then for all elements Z such that $X \prec Z \prec Y$ it holds that $X \subseteq Z \subseteq Y$.*

Firstly, we show that $Z \prec Y$ and $Z \nsubseteq Y$ implies $Z \prec X$ and secondly, that $X \prec Z$ and $X \nsubseteq Z$ implies that $Y \prec Z$. Sets $Z \prec Y$ such that $Z \nsubseteq Y$ require that $Y \leqslant_l Z$. Since $X \leqslant_l Y$, then $X \leqslant_l Z$ and thus $Z \prec X$. Dually, sets $X \prec Z$ such that $X \nsubseteq Z$ require that $Z \leqslant_l X \leqslant_l Y$ and thus, $Y \prec Z$.

Algorithm 1: PreviousClosure algorithm

Input: Stem base $\mathcal{L}$, $candidates$ ordered w.r.t. $\prec$
Output: Pre-closure $\mathcal{L}^\bullet(X)$ or $null$ when enumeration is over, and updated list of $candidates$

```
for X ∈ candidates do
    if X ⩽_l L•(X) then
        candidates = {Y ∈ candidates | L•(X) ≺ Y} return L•(X)
    end
end
return null
```

Optimizations for Algorithm 1 are not described for the sake of simplicity. However, we mention that the set of candidates is dynamically built.

Algorithm 2 presents the mechanisms used to calculate the stem base $\mathcal{L}$. The algorithm receives the partition pattern structure as an input and initializes an empty stem base and a list of lectically ordered candidates (powerset of attributes). Then, it calls the *PreviousClosure* procedure defined in Algorithm 1 until a *null* value is returned. Once a pre-closure is found, the algorithm tests if it is not a closure (line 5) which means that it is a pseudo-closure. In such a case, a new implication is added to the stem base (line 6) and the algorithms iterates.

Notice that the correct calculation of pre-closures is secured by the lectical order of the candidates and the fact that $P \subseteq X \implies P \prec X$. Thus, in Algorithm 1, we are certain that for calculating $\mathcal{L}^\bullet(X)$, we have all $P \subset X$ required by Proposition 3.

Algorithm 2: Stem Base Miner Algorithm

Input: Partition Pattern Structure $(\mathcal{U}, (D, \sqcap), \delta)$
Output: Stem base $\mathcal{L}$

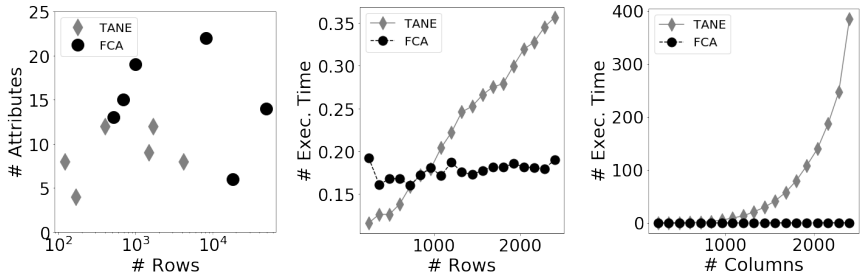
```
1 L = ∅
2 candidates = (℘(U), ≺)
3 X = previous_closure(L, candidates)
4 while X is not null do
5 |   if X ≠ X^□□ then
6 |   |   L = L ∪ {X → X^□□}
7 |   end
8 |   X = PreviousClosure(L, candidates)
9 end
```

For example, consider the database in Table 1 with 5 attributes. The set of candidates contain $2^5 = 32$ elements to be tested. The first candidate is e. At this point $\mathcal{L}$ is empty, so $\mathcal{L}^\bullet(e) = e$ is a pre-closure. Since $e^{\square\square} = e$, no rules are added to the stem base. For the candidate d we have that $\mathcal{L}^\bullet(d) = d$ and $d^{\square\square} = de$, meaning that the pre-closure d is pseudo-closed. Thus, rule $d \rightarrow e$ is added to $\mathcal{L}$. As an example of pruning, consider candidate cd. Since $\mathcal{L} = \{d \rightarrow e\}$, we have $\mathcal{L}^\bullet(cd) = cde$. Line 3 of Algorithm 1 removes from the set of candidates cde and the next candidate is b.

5 Experimental Evaluation

We compare our approach to TANE [14], a state-of-the-art FD miner. TANE is a highly optimized *apriori*-based algorithm that generates a canonical cover of FDs. A canonical cover usually contains many more dependencies than those in a minimum cover such as the one we are able to mine using our approach. Nevertheless, a canonical cover can be reduced to a minimum cover using an algorithm that has quadratical complexity w.r.t. its cardinality [11].

The goal of this evaluation is to study the comparative benefits of using our approach versus a traditional approach such as TANE. TANE was re-implemented for



(a) Datasets and the number of rows and columns they contain.

(b) Performance Comparison when increasing tuples: FCA vs TANE

(c) Performance Comparison when increasing attributes: FCA vs TANE

Dataset	# Columns	# Rows	FCA # CC Deps	FCA Time [S]	FCA/PS # MC Deps	FCA/PS Time [S]	TANE # CC Deps	TANE Time [S]
Mushroom	8124	22	3605	28380	1509	13559.9	-	-
Adult	48842	14	78	110.07	42	138.82	78	305.59
Credit	690	15	1099	3.29	253	2.40	1099	6.21
PGLW	17995	6	5	0.58	2	0.35	5	1.45
PGLW (2xA)	17995	6	38	1.53	15	1.21	38	19.55
Forest Fires	516	13	442	0.75	138	0.65	442	0.66
Forest Fires (2xT)	516	13	442	3.05	138	3.16	442	5.92
ncvoter	19	1000	775	3.01	193	2.68	775	5.06
Diagnostics	120	8	37	0.14	17	0.15	37	0.08
Abalone	4177	8	137	2.2	40	2.40	137	0.41
CMC	1473	9	1	0.85	1	0.90	1	1.02
Hughes	401	12	3	0.19	3	0.19	3	0.12
Servo	167	4	1	0.1	c1	0.11	1	0.06
Caulkins	1685	12	227	1.99	67	2.16	227	1.76

Table 4: Dataset details, Execution Times in Seconds, and Number of Mined Rules for FCA, FCA/Pattern Structures and TANE. CC: Canonical Cover, MC: Minimum Cover. Datasets in boldface represent those in which FCA performed better than TANE.

our experiments[4]. This implementation and our approach were coded in Python. The software is freely available[5].

We performed experiments over 12 datasets extracted from the UCI Machine Learning repository[6], the JASA Data Archive[7] and Metanome's repeatability Web page[8]. Details on the number of rows and columns for each dataset are provided in the first two columns of Table 4. In addition to these datasets, we created synthetic versions by multiplying the rows or the columns of a given dataset. Experiments were run over an Intel Core i7 running at 2.2 Ghz and equipped with 16 GB of RAM memory.

[4] https://github.com/anonexp/fudep

[5] https://github.com/anonexp/fd_miner

[6] https://archive.ics.uci.edu/ml/index.php

[7] http://lib.stat.cmu.edu/jasadata/

[8] https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html

5.1 Results & Discussion

Table 4 presents the main results of applying our approach and TANE on each dataset to mine the Minimum Cover and the Canonical Cover, respectively. The table contains the execution times of each algorithm and the number of dependencies mined. Datasets in boldface represent those for which FCA performed better than TANE. For the Mushroom dataset, TANE was not able to obtain results before running out of memory, thus no information is provided in the table. Table 4 also reports in two synthetic datasets, namely PGLW (2x Attributes) which contains two horizontal copies of the PGLW dataset resulting in twice as many attributes. Forest Fires (2x Tuples) contains two vertical copies of Forest Fires resulting in twice as many tuples. All mined Canonical Covers mined by TANE have been reduced to a Minimum Cover to verify the consistency of our approach.

Out of the 12 datasets, our approach performs better in the **largest** (both in rows and columns). This is better depicted by Figure 1a where datasets are represented as points in a rows-columns space. Circles represent datasets for which our approach performed better while diamonds, where TANE did. Notice that the X axis is provided in logarithmic scale. The figure shows that most of the datasets where TANE performs better are in the lower-left region of the figure, representing small datasets. Our approach performs better in datasets on all other regions, including the upper-right which contains datasets with many rows and columns.

Synthetic datasets in Table 4 show evidence that our approach scales better when duplicating the dataset. When duplicating attributes the difference is particularly dramatic since TANE is over 13 times slower while our approach, only 3. To study this further, we created two sets of 19 synthetic datasets. The first set (Vertical set) incrementally multiplied vertically the *Diagnostics* datasets (with 8 attributes and 120 tuples) generating versions of 240, 360, 480 tuples, and so on up to a dataset containing 2400 tuples. The second set (horizontal set) of datasets did the same in a horizontal manner generating versions of 16, 24, 32, up to 160 attributes. Since most of the datasets of the second set were too big for TANE, they were trunked to just 40 tuples.

Figure 1b depicts the increasing time for TANE and FCA on the vertical set, i.e. when increasing the number of tuples. We can observe that both approaches scale linearly w.r.t. the number of tuples, even when our approach seems to have a much more stable behavior. Vertical multiplication of datasets yield the same number of FDs than the original set, since the relation between attributes remains unchanged. Thus, we can assume that other algorithms based on TANE could achieve a similar performance than our approach provided some optimizations.

On the other hand, this may not to be the case for the horizontal set. Figure 1c shows that our approach remains very stable when varying the number of attributes, while TANE's execution time grows exponentially. Indeed, this great difference in performance is due to the way in which our approach finds FDs which differs from TANE's strategy. Consider the synthetic dataset with many copies of a given attribute. In this case, the pruning mechanism described in Section 4.3 quickly removes thousands of candidates by means of the closure operator. Instead, TANE computes each attribute combination rendering the exponential growth in the computation time.

We stress that this is not simply an extreme case from which our approach takes advantage, but actually a very good illustration of the benefits of using a closure operator to navigate the space of FDs. Closures enable our approach to avoid unnecessary computations not only when we have redundant attributes, but also whenever possible in the lattice of the powerset of attributes.

6 Conclusions

We have presented a new characterization of a minimum cover of functional dependencies (FDs) by means of the stem base (or Duquenne-Guigues basis) of a partition pattern structure. We have presented the mechanisms through which this characterization can be exploited to efficiently mine the minimum cover. Furthermore, we have described the algorithms that implement these mechanisms and show empirical evidence that our characterization performs better than a state-of-the-art FD miner, namely TANE, in larger databases containing many rows and columns.

Acknowledgments. This research work has been supported by the SGR2014-890 (MACDA) project of the Generalitat de Catalunya, and MINECO project APCOM (TIN2014-57226-P) and partially funded by the French National Project FUI AAP 14 Tracaverre 2012-2016.

References

1. Mehwish Alam, Aleksey Buzmakov, Víctor Codocedo, and Amedeo Napoli. Mining definitions from RDF annotations using formal concept analysis. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 823–829. AAAI Press, 2015.
2. Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Computing Functional Dependencies with Pattern Structures. In Laszlo Szathmary and Uta Priss, editors, *CLA*, volume 972 of *CEUR Workshop Proceedings*, pages 175–186. CEUR-WS.org, 2012.
3. Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Characterizing Functional Dependencies in Formal Concept Analysis with Pattern Structures. *Annals of Mathematics and Artificial Intelligence*, 72(1–2):129–149, October 2014.
4. Karell Bertet and Bernard Monjardet. The multiple facets of the canonical direct unit implicational basis. *Theoretical Computer Science*, 411(22-24):2155–2166, 2010.
5. Víctor Codocedo, Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Characterization of Order-like Dependencies with Formal Concept Analysis. In Marianne Huchard and Sergei Kuznetsov, editors, *Proceedings of the Thirteenth International Conference on Concept Lattices and Their Applications, Moscow, Russia, July 18-22, 2016.*, volume 1624 of *CEUR Workshop Proceedings*, pages 123–134. CEUR-WS.org, 2016.
6. Víctor Codocedo and Amedeo Napoli. Lattice-based biclustering using Partition Pattern Structures. In *Proceedings of ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 213–218. IOS Press, 2014.
7. Wenfei Fan. Dependencies revisited for improving data quality. In Maurizio Lenzerini and Domenico Lembo, editors, *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pages 159–170. ACM, 2008.

8. Wenfei Fan. Data quality: From theory to practice. *SIGMOD Record*, 44(3):7–18, 2015.
9. Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
10. Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.
11. Benhard Ganter and Sergei Obiedkov. *Conceptual Exploration*. Springer, Berlin, 2016.
12. Benhard Ganter and Rudolph Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.
13. Jean-Louis Guigues and Vincent Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines*, 95:5–18, 1986.
14. Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Computer Journal*, 42(2):100–111, 1999.
15. Sergei O. Kuznetsov. Galois Connections in Data Analysis: Contributions from the Soviet Era and Modern Russian Research. In Benrhard Ganter, Gerd Stumme, and Rudolf Wille, editors, *Formal Concept Analysis, Foundations and Applications*, Lecture Notes in Computer Science 3626, pages 196–225. Springer, 2005.
16. Sergei O Kuznetsov and Sergei A Obiedkov. Comparing Performance of Algorithms for Generating Concept Lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14:189–216, 2002.
17. Heikki Mannila and Kari-Jouko Räihä. *The Design of Relational Databases*. Addison-Wesley, Reading (MA), USA, 1992.
18. Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proc. VLDB Endow.*, 8(10):1082–1093, 2015.
19. Uwe Ryssel, Felix Distel, and Daniel Borchmann. Fast algorithms for implication bases and attribute exploration using proper premises. *Annals of Mathematics and Artificial Intelligence*, 70(1):25–53, feb 2014.
20. J.D. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.
21. Yang Yu and Jeff Heflin. Extending functional dependency to detect abnormal data in RDF graphs. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 794–809. Springer, 2011.
22. Yang Yu, Yingjie Li, and Jeff Heflin. Detecting abnormal semantic web data using semantic dependency. In *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*, pages 154–157. IEEE Computer Society, 2011.